Energy-Efficient Monitoring of Spatial Predicates over Moving Objects

Haibo Hu   Dik Lun Lee
Department of Computer Science
Hong Kong University of Science and Technology
{haibo,dlee}@cs.ust.hk

Abstract

We study the problem of monitoring the changes of predicates on the spatial relations between objects, which is fundamental for many monitoring and tracking applications. As the location update problem is the core of object monitoring, we develop a cost model on the power consumption and propose a family of energy-efficient location update schemes. In these schemes, upon each location update the server sends back additional information about the predicates so that the client can defer the next update until it is about to change the predicate value. As a result, the updates are less frequent than the traditional periodic or dead-reckoning schemes without undermining the monitoring accuracy.

1 Introduction

With the flourishing of location and tracking sensors, monitoring the positions of moving objects becomes a reality. This paves the way for the mass deployment of all kinds of location-aware applications. However, the problem of when or how often the location sensors should report the locations of the objects they are monitoring or tracking has been ignored. These reporting activities are known as “location updates”. A large number of location updates not only incur huge network traffic, but pose heavy burden on the servers that manage the location data. In addition, frequent location updates exhaust the sensor power supply quickly, which is a serious issue in mobile and sensor networks. On the other hand, an insufficient number of location updates lead to severe monitoring error.

In the literature, the predominant location update paradigm is the “periodic” scheme, which schedules the updates at regular intervals. However, this scheme cannot scale to large populations and the optimal interval values must be tuned for different systems. It is also inefficient in terms of its monitoring accuracy versus the number of location updates it incurs. In this paper, we first propose the problem of monitoring spatial predicates. These spatial predicates, aligned with those defined in the Open Geospatial Consortium reference model [3], are defined as Boolean assertions on the topological, metric, or directional relations between spatial objects. We then develop the cost model of power consumption for this monitoring problem and propose a family of location update schemes that reduces this cost. As the dynamics of these spatial predicates are interesting to many location-aware monitoring services (e.g. fleet tracking, combat troop tactics) and are also regarded

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

as the primitives of spatial queries, these schemes serve as a general solution to a wide range of monitoring applications. These schemes take the same approach that, along with each location update the server sends back additional information (more specifically the *safe regions*) about the spatial predicates being monitored so that the client can delay the next update until it is about to change the predicate value. In other words, these schemes achieve fewer number of updates without undermining the monitoring accuracy.

The rest of the paper is organized as follows. Section 2 formally defines the spatial predicates and relates them to spatial queries. Section 3 proposes the notion of safe regions and the operations on them. Following the system model in Section 4, we develops the cost model of power consumption in Section 5 based on which the family of update schemes are presented.

2 Spatial Predicates

Definition 1: A **spatial predicate** is a Boolean assertion on the topological, metric, or directional relations between two spatial entities a and b. It is denoted as $a \circ b$, where $\circ$ stands for one of the relations in the following categories:
1. topological relations: “contains” ($\odot$), “overlap” ($\otimes$), “disjoint” ($\oslash$), “within” ($\ominus$), etc;
2. metric relations: “nearest to” ($\Rsh$), ”farthest to” ($\dashrightarrow$), “neighboring” ($\curvearrowright$), etc;
3. directional relations: “north of” ($\trianglelefteq$), “south of” ($\trianglerighteq$).
Furthermore, a is called the *subject* and b is called the *observer* of the predicate.

The subject corresponds to a moving object, whereas the observer could be either moving or stationary. By this definition, all conventional spatial queries can be expressed as a combination of these spatial predicates. For example, a nearest neighbor query corresponds to some “nearest to” predicates in which the query point is the observer and a neighborhood object is the subject; a window query corresponds to some “within” and “overlap” predicates in which the query window is the observer; and a spatial join between two datasets corresponds to the predicates in which each object in the second dataset serves as the observer. The formal correspondence of these queries to the spatial predicates on the moving object dataset $\mathcal{S}$ are listed below.

Example 1: 1) Range query with range q: $Range(q) = \{s \in \mathcal{S} | s \curvearrowright q = true\, AND\, (s \ominus q \vee s \otimes q) = true\}$
2) Nearest neighbor query with query point (or query object) q: $NN(q) = \{s \in \mathcal{S} | s \curvearrowright q = true\, AND\, s \Rsh q = true\}$
3) Distance spatial join for $\mathcal{S}$ and another dataset $\mathcal{R}$ within distance threshold δ: $Dist\_Join(\mathcal{S}, \mathcal{R}, \delta) = \{(s,r) | (s \ominus (r \bigoplus \delta) \vee s \otimes (r \boxplus \delta)) = true, r \in \mathcal{R}, s \in \mathcal{S}\}$, where $\boxplus$ stands for the Minkowski sum [4].

As such, continuously monitoring these spatial queries is equivalent to monitoring the changes of the corresponding spatial predicates. Besides spatial queries, many location-aware applications require the notification of certain spatial events, which can be expressed as spatial predicates. The following is a typical example.

Example 2: In a mobile ad-hoc network, mobile client s is the parent of client a, b and c in the routing tree (see Figure 1). In the physical transmission layer, s must reach a, b and c. As such, in case any child moves beyond the effective transmission range δ of s, the routing topology of a, b, c and s must be rebuilt. Monitoring this event is equivalent to monitoring the spatial predicates $i \ominus (s \bigoplus \delta)$ for each $i \in \{a, b, c\}$. The event is triggered when any of these predicates becomes false.

3 Safe Region

The notion of *safe region* has been explored in the area of moving object monitoring [6, 8]. We apply the same notion but provide a set of formal definitions and operations on the safe regions. To simplify the presentation, we assume that a moving object is represented by a point.

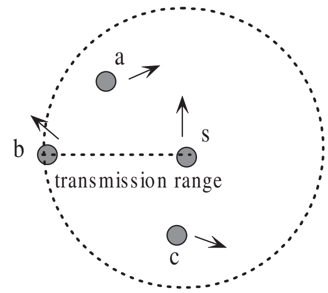
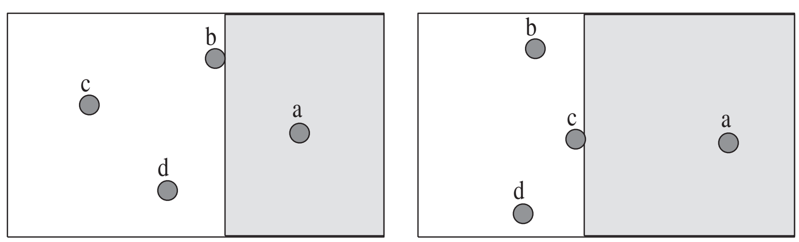


Figure 1: An Example of Monitoring Spatial Events in A Mobile Ad-Hoc Network

Definition 2: The *safe region* of a moving object a at time t_0 with respect to the spatial predicate $a \circ b$, denoted as $SR(a \circ b, t_0)$, is defined as the largest connected region within which $a \circ b$ does not change its value, i.e., $\forall s \in SR(a \circ b, t_0)$, $s \circ b = a(t_0) \circ b$, where $a(t_0)$ denotes the location of a at time t_0.

The movement of b or other moving objects may change $SR(a \circ b, t_0)$ (e.g., another object becomes "nearest to" b). Figures 2(a) and (b) show the safe regions (the shadowed areas) of object a with respect to the spatial predicate "easternmost" at time t_0 and t_1.



(a) At Time t_0 (b) At Time t_1

Figure 2: $SR(a \circ E)$ at t_0 and t_1

3.1 Operations on Safe Regions

In this subsection, we introduce the set of operations on safe regions, namely, computation, approximation, and combination. In the sequel, we focus on the "within" ($\ominus$) predicate only, but the same rationale applies to the other predicates.

3.1.1 Computation

For $a \ominus b$, if the observer b is a stationary region, $SR(a \ominus b, t_0) = b$ if $a(t_0) \ominus b = true$; otherwise $SR(a \ominus b, t_0) = D - b$, where D is the universe of the space. However, if b moves over time, $SR(a \ominus b, t_0)$ is a function of b's location. To decouple its dependence on b's movement, let ϕ_b denote the maximum velocity of b. We have the following theorem:

Theorem 3: If $a(t_0) \ominus b(t_0) = true$, $SR(a \ominus b, t_0) \subseteq b(t_0) \boxminus \mathcal{C}_{\phi_b \cdot t}$; otherwise $SR(a \ominus b, t_0) \subseteq D - (b(t_0) \boxplus \mathcal{C}_{\phi_b \cdot t})$, where $\mathcal{C}_{\phi_b \cdot t}$ denotes the circle centered at the origin and with radius $\phi_b \cdot t$, and $\boxminus$ is the Minkowski subtraction.

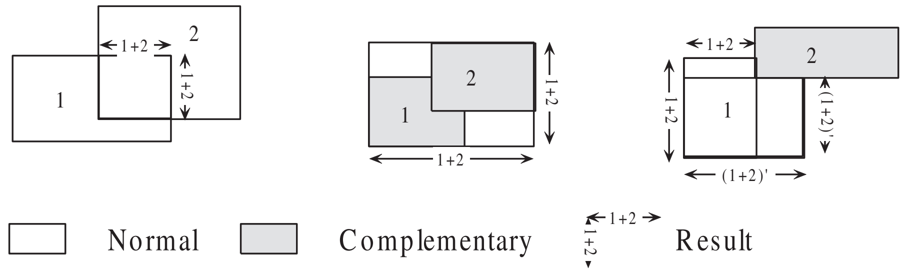


Figure 3: Three Cases of Combining Two Bounded Boxes

3.1.2 Approximation

To reduce the storage and computational cost of an arbitrary-shaped safe region, we approximate the arbitrary shape with a shrinking rectangle (or the complement of an expanding rectangle) with respect to the elapsed time t. The rectangle is called the *bounded box* of the safe region, and is denoted by $BB(a \circ b, t_0)$. At any time t, $BB(a \circ b, t_0) \subseteq SR(a \circ b, t_0)$. The bounded box is organized as follows:

Definition 4: Let the 4-tuple $\langle x_{low}, x_{high}, y_{low}, y_{high} \rangle$ denote a rectangle $\mathbf{R}$, with the tuple elements representing the four bounding edges of $\mathbf{R}$. $BB(a \circ b, t_0)$ is a 3-tuple: $\langle complement, \mathbf{R}_{t_0}, \mathbf{V} \rangle$, where $\mathbf{R}_{t_0}$ is the initial bounded box at t_0, $\mathbf{V}$ is its shrinking/expanding velocity on the four edges, and $complement$ is a Boolean designating whether the bounded box is shrinking or expanding.

To compute the approximation for $BB(a \ominus b, t_0)$, we replace $\mathcal{C}$ with its minimum bounding box in Theorem 3, and then compute the Minkowski sum or subtraction.

3.1.3 Combination

The bounded boxes of two safe regions, $BB(a \circ b, t_0)$ and $BB(a \circ c, t_0)$, can be combined (denoted as "$+$") to form a new bounded box in which the movement of object a does not change the values of both spatial predicates. The combination can be extended to more than two safe regions, which is denoted as "$\vee$".

To decide the $complement$ value of the combined box, we distinguish three cases: (1) combining two normal boxes, (2) combining two complementary boxes, and (3) a normal box combining a complementary one. Figure 3 illustrates the operation. For case (1), the combination is a normal box, and its $\mathbf{R}_{t_0}$ is the intersection of the two original $\mathbf{R}_{t_0}$'s, and its $\mathbf{V}$ is the maximum of the two original $\mathbf{V}$'s. For case (2), the combination is complementary, and its $\mathbf{R}_{t_0}$ is the minimum bounding box of the two original $\mathbf{R}_{t_0}$'s, and its $\mathbf{V}$ is the maximum of the two original $\mathbf{V}$'s.[1] For case (3), the result is defined as a normal box, and its $\mathbf{R}_{t_0}$ is the difference between the normal's and the complementary's $\mathbf{R}_{t_0}$'s,[2], the speed of its $\mathbf{V}$ is the maximum speed of the two original $\mathbf{V}$'s, and the direction of its $\mathbf{V}$ is opposite to that of the original normal $\mathbf{V}$.

4 The Abstract System Model

In this section, we describe the system model under which our location update schemes are developed. The model is abstract in that it makes no assumption on the underlying network infrastructure. Figure 4 shows the relations between the mobile clients (MC), the location server (LS) and the location-aware services (LAS): the LASs register the spatial predicates at the LS; and the MCs update their locations to the LS, which reports to the LASs when the monitored values change.

[1] By maximum, we mean the velocity that has the maximum speed.

[2] If the difference is not a rectangle, $\mathbf{R}_{t_0}$ is defined as the maximum bounded box of the difference.

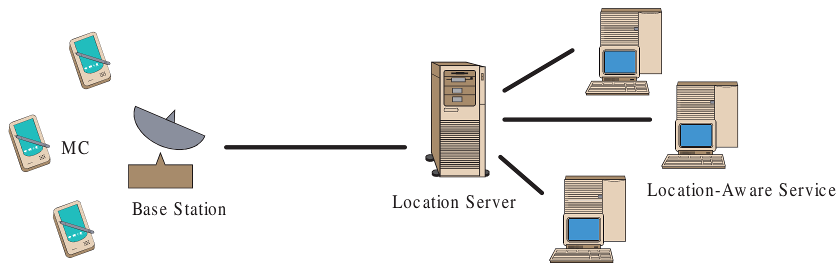


Figure 4: The Abstract System Architecture

We make the following assumptions and notations on the MC's behavior:

- The MC measures its location and determines the necessity of a location update every t_m unit of time.
- The location update is sent through an uplink channel, which costs a constant power consumption $cost_u$.
- The LS then sends to the MC through a downlink channel the approximated and combined bounded boxes for the safe regions with respect to all the spatial predicates being monitored.[3] The power consumption to receive a bounded box is $cost_d$.
- The power consumption for the MC to test if it is inside a bounded box is $cost_c$. As such, if n bounded boxes are sent to the MC, it needs $n \cdot cost_c$ in every t_m time units to make the location update decision.

5 Energy-Efficient Update Schemes

Conventional periodic or distance-based location update schemes [7] are not optimal in terms of update frequency. Better schemes should take into account the locality of the monitored spatial predicates so that the clients perform updates only when some predicate value might change. In its simplest form, the server sends back to the client the safe region of each predicate with respect to its current location so that the MC can check locally if its movement changes the value of some predicate. However, this approach incurs significant bandwidth and CPU overhead as the client has to test if it moves out of any safe region. In what follows, we present a family of energy-efficient schemes that *approximate* the safe regions with their bounded boxes, and they differ in their ways of combining these boxes.

More specifically, we first compute the safe region and then the bounded box of each predicate. All the normal bounded boxes are simply combined to form a single normal box BB_*. The proposed schemes differ in how they treat the complementary boxes. In short, a complementary box BB_c can be either discarded (if $BB_* + BB_c = BB_*$), combined with BB_*, left alone, or combined with other complementary boxes. The tradeoff is the accuracy of the approximation and the power consumption in receiving and testing these boxes. The proposed schemes aim to minimize the *average power consumption rate* before the next location update. We target the power consumption not only because this is one of the major issues in mobile and pervasive computing, but also because the reduction of location update cost in terms of power consumption also addresses other major issues such as bandwidth and response time. In the following, we first develop the power consumption model and then propose the schemes.

[3]There are three reasons why the safe region information is sent as the LS's response to the MC's location update. First, this is the only time when the LS has the exact knowledge about the MC's current location, so the safe region(s) best suits MC's need. Second, we do not assume a broadcast channel in this architecture because the power consumption of listening on this channel is forbiddenly high. Third, a request-response communication incurs less connection overhead than two independent one-way communications.

5.1 The Power Consumption Model for Mobile Clients

Let n denote the number of bounded boxes sent to the client and $\overline{t_d}$ the average elapsed time until the next location update. The average power consumption rate before the next location update, denoted as $\overline{cost_t}$, is defined as:

$$\overline{cost_t} = \frac{cost_u + n \cdot cost_d + n \cdot cost_c * \frac{\overline{t_d}}{t_m}}{\overline{t_d}} = \frac{cost_u + n \cdot cost_d}{\overline{t_d}} + \frac{n \cdot cost_c}{t_m}, \tag{1}$$

where t_m, $cost_c$, $cost_d$ and $cost_u$ are constants as defined in Section 4.

To derive $\overline{t_d}$, we make the following assumptions on the objects' movement:

- The object can freely choose a random moving direction and speed.
- It does not change the direction and speed until the next location update.
- The average speed is $\overline{\phi}$.

In other words, a moves along a straight line starting from its current updated location $a(t_0)$. On the other hand, $\overline{t_d}$ also depends on the bounded boxes, which change with time. Thus, at any given time t, the average elapsed time until the next update $\overline{t_d}(t)$ is:

$$\overline{t_d}(t) = \frac{\int_0^{2\pi} |kd_t(\theta)a(t)|d\theta}{2\pi\overline{\phi}}, \tag{2}$$

where θ is the angle between the moving direction and the positive x-axis, kd_t is the first intersection point of this line with the boundary of $BB_*(t)$ and all $BB_c(t)$. Obviously $\overline{t_d}$ is a solution to the following equation of t:

$$t = \overline{t_d}(t) \tag{3}$$

To solve $\overline{t_d}$ efficiently using Equation 3, we propose an approximation of $\overline{t_d}(t)$.

Lemma 5: Given a connected region R and an inner point p, if for any θ, the line from p with an angle θ to the x-axis intersects R's boundary only once (the intersection is denoted as $kd(\theta)$), then

$$\int_0^{2\pi} |kd(\theta)p|d\theta \approx \frac{2 \cdot Area(R)}{\delta(R)},$$

where $\delta(R)$ is the average distance between two points in R.

We replace p with $a(t)$ and replace R with $BB_*(t) - \vee_c BB_c(t)$ in the lemma, and obtain the following approximation of $\overline{t_d}(t)$ from Equation 2:

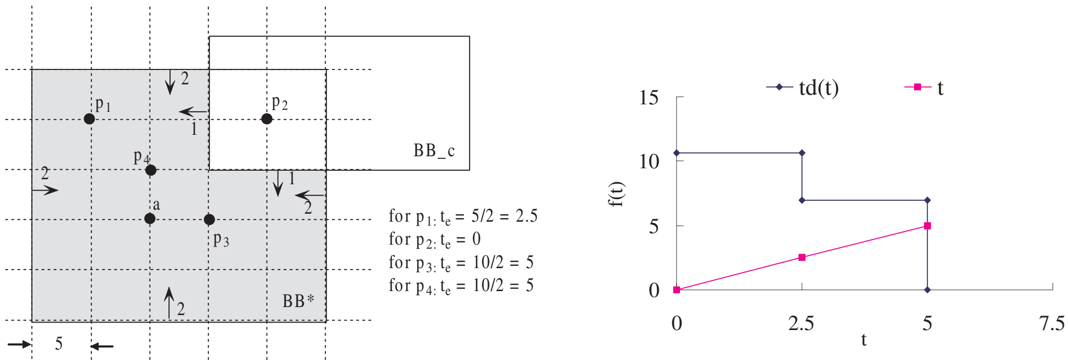
Theorem 6:

$$\overline{t_d}(t) \approx \frac{Area(BB_*(t) - \vee_c BB_c(t))}{\pi\overline{\phi} \cdot \delta(BB_*(t) - \vee_c BB_c(t))}$$

To compute $\overline{t_d}(t)$ in Theorem 6, we apply the Monte Carlo method to compute $Area$ and δ of $BB_*(t) - \vee_c BB_c(t)$. First, we randomly pick K fixed points in $BB_*.\mathbf{R}_{t_0}$ as seeds. For each seed p, let $p.t_e[i]$ denote the elapsed time from t_0 to the time when p becomes outside an individual BB_* or BB_c box i, and $p.t_e$ denote the smallest value $p.t_e[i]$ among all i. As such, $p.t_e$ is essentially the elapsed time from t_0 to the time when p leaves $BB_*(t) - \vee_c BB_c(t)$.

Each seed p accounts for an area of size $Area(BB_*.\mathbf{R}_{t_0})/K$. p contributes this portion of area to the area of $BB_*(t) - \vee_c BB_c(t)$ only when $t \leq p.t_e$. Similarly, $\delta(BB_*(t) - \vee_c BB_c(t)) = 2\sum_{p,p'} |pp'|/K(K-1)$ and each p contributes to δ only when $t \leq p.t_e$. As t increases, more and more p's stop contributing to $Area$ and δ. Therefore $\frac{Area(BB_*(t) - \vee_c BB_c(t))}{\pi\overline{\phi} \cdot \delta(BB_*(t) - \vee_c BB_c(t))}$, i.e., $\overline{t_d}(t)$, is a staircase function. As $\overline{t_d}$ is the solution of Equation 3, it is the intersection point of functions $\overline{t_d}(t)$ and t.

An example of this method is illustrated in Figure 5. Figure 5(a) shows the BB_*, BB_c and the four seeds p_1 through p_4, and the grey area is $BB_*(t_0) - \vee_c BB_c(t_0)$. Figure 5(b) shows the corresponding $\overline{t_d}(t)$ function computed through Theorem 6. $\overline{t_d}$ is its intersection point with function t, which is 5.



(a) The Setting

(b) The Staircase Area Function

Figure 5: The Monte Carlo Method to Compute $\overline{t_d}$

5.2 The Greedy Split Scheme (GSS)

From Equation 1, minimizing $\overline{cost_t}$ is equivalent to finding a tradeoff between n, the number of BB_c's sent to the client and $\overline{t_d}$. At one extreme, all BB_c's can be sent individually (i.e., maximizing n) while at the other extreme, all of them can be combined into one BB_c (i.e., minimizing n). The family of update schemes takes a moderate approach and divides BB_c's into groups. Each group of BB_c's is combined into one BB_c and sent to the client. In this and the next subsections, we present three of the family members with different grouping algorithms.

The GSS scheme initially considers all BB_c's as one group and greedily divides it into subgroups. The scheme applies the nodes splitting algorithms in R-tree or R*-tree [5, 1]. The splitting of a group continues until it contains only one box or a further split cannot reduce $\overline{cost_t}$.

As for the time complexity, each splitting takes linear or quadratic time (depending on the splitting algorithm) of C, the number of BB_c's. It also requires $O(K)$ to compute the $\overline{t_d}$ with an incremental Monte Carlo method. As the maximum number of splitting is at most $C - 1$, the overall time complexity of the GSS scheme is $O(C(C + K))$ for the linear splitting algorithm and $O(C(C^2 + K))$ for the quadratic algorithm.

5.3 The Greedy Sort–Merge Scheme (GSM)

The GSM scheme first sorts all the BB_c's by some "Space Filling Curve" [2] according to their spatial locality. Then it linearly scans from the first box ($BB_c[0]$) to test if it should combine with the next box ($BB_c[1]$). The two boxes are combined if the $\overline{cost_t}$ for the combined box is smaller; otherwise $BB_c[1]$ starts a new group.

As for time complexity, the initial Monte Carlo method to compute $\overline{t_d}$ takes $O(K(C + K))$ time. Further, each test on combining involves an incremental Monte Carlo method that costs $O(K)$ time. Since the maximum number of such tests is $C - 1$, the overall time complexity of the GSM scheme is $O(K(C + K))$.

5.4 The p-Means Clustering Scheme (PMC)

The p-Means clustering scheme (PMC) applies a clustering approach: it first chooses some centroid BB_c's as the initial clusters. Then it tests each of the remaining BB_c's if it should be combined with the nearest BB_c cluster or should become a new cluster. The initial set of clusters should be chosen to be the dominant part of $\overline{t_d}$ so that the scheme is stable. From the Monte Carlo method, the more p's whose $p.t_e$ values are determined by a

BB_c, the more significant this BB_c is in contributing to $\overline{t_d}$. As such, we denote the *significance* of a BB_c as the number of such p's and the initial BB_c clusters are the most significant boxes.

As for time complexity, the selection of the initial BB_c clusters takes $O(K(C+K)+ClogC)$ time. In the worst case, the test on each of the remaining BB_c's needs $O(C+K)$ time. Therefore, the overall time complexity is $O(C(C+K)+K^2)$.

6 Conclusion and Vision

To minimize the power consumption cost of location updates, we proposed a family of update schemes for spatial predicate monitoring. Our contributions are:

1. We proposed the problem of spatial predicate monitoring and showed its correspondence to spatial query monitoring.
2. We developed a precise cost model of power consumption for the monitoring problem.
3. We proposed a family of location update schemes that reduce the power consumption based on the cost model.

We foresee the ever-increasing demand of moving object monitoring, and in particular monitoring techniques that can tolerate uncertain or imprecise results. The tradeoff between the precision and the cost of the monitoring is critical in mobile ad-hoc and sensor networks and calls for more research efforts. We also envision the mass deployment of dedicated (moving) location sensors that can form ad-hoc networks and monitor other moving objects. This leads to a more challenging monitoring problem in which various factors such as the sensor movement, the sensing frequency and the update frequency should be considered.

References

[1] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *ACM SIGMOD Conference*, pages 322–331, 1990.

[2] T. Bially. Space-filling curves: Their generation and their application to bandwidth reduction. *IEEE Transactions on Information Theory*, 15(6):658–664, 1969.

[3] Open Geospatial Consortium. OGC reference model. http://www.opengeospatial.org/specs/?page=orm.

[4] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 1997.

[5] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *ACM SIGMOD Conference*, pages 47–57, 1984.

[6] H. Hu, J. Xu, and D. Lee. A generic framework for monitoring continuous spatial queries over moving objects. In *ACM SIGMOD Conference, Baltimore, Maryland*, pages 479–490, 2005.

[7] V. Kumar and S. R. Das. Performance of dead reckoning-based location service for mobile ad hoc networks. *Wireless Communications and Mobile Computing Journal*, 4(2), March 2004.

[8] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE Transactions on Computers*, 51(10):1124–1140, 2002.